White Paper: Developing a Dynamic Performance Information Infrastructure for Grid Systems

Rich Wolski[*]
University of Tennessee

Martin Swany[†]
University of Tennessee

Steven Fitzgerald[‡]
University of Southern California
California State University, Northridge

February 9, 2000

Abstract *This document specifies a potential architecture for managing dynamic performance information in Computational Grid settings. The architecture's components and relationships are motivated by our recent experiences in building dynamic performance information infrastructures for a variety of Grid systems.*

1 Introduction

Computational Grids require timely and accurate dynamic performance information for two reasons. First, dynamic application and resource schedulers must base their decisions on what available performance will be available. Predicting available performance from static estimates does not allow schedulers to consider the dynamically changing performance conditions. Secondly, dynamic performance information is necessary to implement fault recognition and diagnosis. Often, component failures are soft (particularly in the network where alternative routing may partially mask a problem) so faults manifest themselves as reductions in delivered performance.

The purpose of this document is to outline an architecture for the management and control of dynamic performance information in Computational Grid settings. Several characteristics distinguish the problem of managing dynamic performance information from the provision of other Grid services.

- **Performance information has a fixed, often short lifetime of utility**. Data may go stale quickly making rapid read access important, but obviating the need for long-term storage (except for archival purposes).
- **Updates are frequent.** Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently then it is read. Most extant information-base technologies are optimized for query and not update making them potentially unsuitable for a dynamic information storage.
- **Performance information is often stochastic**. It is frequently impossible to characterize the performance of a resource or application component using a single value. Therefore, dynamic performance information may carry *quality-of-information* metrics quantifying its accuracy, distribution, lifetime, etc. which may need to be calculated from the raw data. These calculations add a computational component to the information management system itself.

[*]rich@cs.utk.edu
[†]swany@cs.utk.edu
[‡]steve@ecs.csun.edu

- **The data gathering and delivery mechanisms must be high-performance**. Because dynamic data may grow stale quickly, the data management system must optimize the elapsed time associated with storage and retrieval. Note that this requirement differentiates the problem of dynamic data management from the problem of providing an archival performance record. The elapsed time to read an archive, while important, is often not the driving design characteristic for the archival system. We believe that archival data will be useful both for accounting purposes and for long-term trend analysis. It is our belief, however, the separate but complimentary systems for managing and archiving Grid performance data respectively are required, each tailored to meet its own set of unique performance constraints.
- **Grid performance measurement facilities must be scalable**. With the potential for thousands of resources and tens-of-thousands of Grid users to be using the Grid simultaneously, it will be important for monitoring facilities to be able to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

It is important to realize that the requirements for dynamic performance information are different than that for archival or static data. Our contention is that separate logical architectures for each type of data are required. These architectures must be able to interoperate, but need not share implementations. We focus on an architecture for gathering and managing dynamic data exclusively in this paper. Also, we hasten to emphasize that this document outlines our observations about the abstractions that have proved relevant to date. We intend it to serve as an invitation for discussion and a catalyst further investigation. Our goal is to understand more fully the requirements for managing dynamic data on the Grid, but not to attempt to mandate a particular implementation strategy.

2 Architecture

The architecture we propose consists of

- a **directory service** for naming and locating performance data and data sensors,
- a scalable and distributed **performance monitoring service** that gathers and stores dynamic performance information,
- a short-term **data repository service** that can store data while it is not stale, and
- an information **caching facility** that allows a user or scheduler to specify a subset of the total information base that is of interest.

According to the tenets of our model, dynamic performance data is gathered by *sensors*. A sensor is any entity that is capable of producing time-stamped performance measurements. We place no restrictions on the data format or units associated with either the time-stamps or the measurements produced by a sensor and assume only that this information is available and can be published in a Grid-accessible information base. Note that this definition is intended to include both resource and application performance sensors as we believe that the management of both information types is critical to Grid performance.

2.1 Directory Service

To locate, name, and describe the structural characteristics of any dynamic data available to the Grid, we believe that a distributed directory service for publishing this information must be available. The primary purpose of this directory service is to allow information consumers (users, programs and resource schedulers, visualization tools) to discover and understand the characteristics of the information that is available. That is, the directory service facilitates *information discovery* within the system.

The directory service, however, is not responsible for the storage of performance data itself – only its name and other characteristics. We assume the names and characteristics associated with dynamic performance

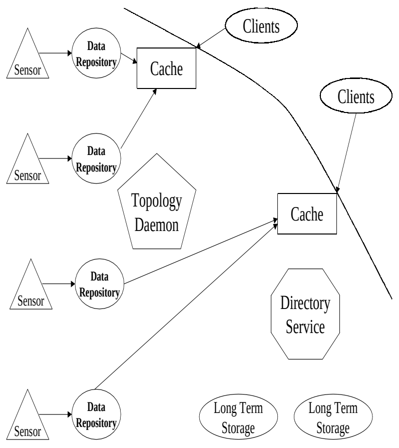

Figure 1: **Architecture Overview**

data will be slowly changing (unlike the data itself). That is, the name and structural characteristics of a data set will remain relatively constant while the valid contents of the data set may change dramatically over time. As such, query-optimized directory services such as the Globus MDS [2], LDAP, the Legion Information Base, Novel NDS, the SDSC Metadata Catalog, etc. provide the necessary base functionality.

Similarly, performance sensors (their location, type, etc.) should be registered with the directory service as, themselves, are Grid resources. Again, unlike the data they gather, we believe that sensor characteristics will be slowly changing making query-optimized directory services appropriate implementation mechanisms.

2.2 Scalable Performance Monitoring

If performance monitors are not coordinated in the Grid, the intrusiveness of performance monitoring may strongly impact available performance, particularly as the system scales. That is, if all performance facilities operate their own sensors, Grid resources will be consumed by the monitoring facilities alone. Coordinating a Grid-wide collection of sensors is complicated by both the scale of the problem (there are many Grid resource characteristics to monitor) and by the dynamically changing performance and availability of Grid resources. We propose that a Grid-wide sensor control facility be developed that

- scales with the Grid,
- allows sensors to join and to leave the system dynamically,
- is able to deliver performance information while it is still valid.

Note that scalability can be provided through a hierarchical structure and some notion of *performance equivalence*. For example, end-to-end network monitoring is critical to effective Grid scheduling. It is not feasible to probe all N^2 links between the N Grid resources as the number of resources is likely to be large. Notice, though, that as resources are separated by greater numbers of shared network gateways, a dominating "bottleneck" gateway is likely to emerge as the performance limiting factor. That is, all hosts at one site (say SDSC) are likely to see the same network performance when communicating with any host at another site (say NCSA) located at a great distance. Only one probe between sites is necessary to characterize the network performance between any pair of hosts located one at each site. This ability to cluster distant resources so that they are represented by a single performance sensor (e.g. as relatively equivalent) naturally lends itself to hierarchy. Hosts at each site probe each other and one "distinguished" host at each site is designated to probe the intersite link.

We have used these techniques successfully to build a scalable end-to-end network performance monitoring facility as part of the NWS which is robust with respect to sensor configuration and high-performance.

2.3 Data Repositories

To address the unique characteristics of performance data, we believe that specialized storage services must be provided. The implementation of the components of such storage services must be designed around the use and characteristics of the performance data. This data is generated from a number of locations and data elements can be generated frequently. It is our belief that separate but complimentary systems for managing and archiving Grid performance data respectively are required — each tailored to meet its own set of unique performance constraints.

Because the data is potentially short-lived, we observe that sensor data can be stored in a distributed set of repositories that are optimized for update and can time-out old data. We believe that sensor data will be written more frequently and in smaller batches than it will be read. Sensors wish to append new measurements to a valid history of existing measurements piecemeal. If old data is timed-out by the repositories, the storage footprint and the update time can be carefully controlled using most existing file systems.

Consequently, a set of distributed repositories that are optimized for update, but do not necessarily support files of nearly indefinite length, is required. For example, practical experience has shown that using a LDAP directory server, which is optimized for read operations, is impractical for data storage. The use of the LDAP protocol, however, makes this data available to a wide variety of LDAP-speaking entities. That is, we want to be able to name and "view" the data as if it has been stored in an LDAP database without burdening the LDAP storage mechanisms with the actual data. As part of our work with the MDS, we have been exploring the use of referral mechanisms that allow the directory service to "call out" to other data management services when a fetch for data is requested. We believe that this decoupling of naming and data storage facilities is critical to the performance of the overall system.

Controlling the "footprint" associated with each repository is important for the same reason that performance sensors should be coordinated – to limit intrusiveness. Since the frequency of data store operations may be high, the data repositories must carefully control the storage resources they consume.

2.4 Data Caches

Often, a data consumer (be it a user or other Grid service) need only access a subset of the total dynamic information base that is available. The naming service, which is optimized for query, may be able to support global searches, but our experience is that data consumers often wish to focus on a data subset based on such searches. We propose that some form of client-specific buffering or caching be interposable between data producers and data consumers.

For example, a data visualization tool may wish to specify a subset of resources to visualize in response to user input. Rather than fetching the data directly from the data repositories (which may be distant from the visualization site) a cached copy may be located near the site where the data is being rendered. The buffer is kept "hot" with the latest performance data for the relevant resources by the caching system. When a rendering is demanded, the copy near the renderer is provided, thereby improving performance. Obviously, the consistency of the copy and the repository are an issue but for many applications, best-effort consistency is sufficient.

Currently, we are using an implementation of the Internet Backplane Protocol [3] (IBP) to implement this caching facility in several of our AppLeS [4, 1] efforts.

Similarly, the data repositories, themselves, may wish to use caching to increase efficiency and decease intrusiveness. As part of our experience with the Network Weather Service (NWS) [5], we find that the overhead associated with writing each individual entry into a file (although it is a circular one) can be quite substantial. Caching the data writes will increase the chance of lost data due to system failure, but will also decease the storage overhead.

3 Issues and Concerns

While we believe that the problem of managing dynamic data on the Grid mandates the design of a separate service architecture, there are several issues that we are deliberately skirting. In particular, we are trying to be somewhat parsimonious and narrow with respect to the scope of this architecture. For example, while it is possible to combine the functionality required to manage archival data dynamic data within one implementation, we believe that architecturally, these are separate functionalities with separate requirements. It must be possible to archive dynamic data, but we envision the archiving to be handled by a separate system (with its own architecture) through a well-defined public interface.

Less clear is the distinction between dynamic performance data and distributed event handling. It is possible to consider simple "events" as having the same characteristics as dynamic performance data. Events, typically, must be processed within a specified time deadline just as the utility of performance data (in dynamically changing Grid systems) has a fixed lifetime. Event information must be propagated according to some consistency model just as some form of data consistency must be ensured for performance data. That is, it is possible to treat each event as an individual performance datum that can be managed by a system tuned for dynamic data management. A key difference, however, between performance data and program events is that an event may, itself, trigger other events while our model performance data is relatively passive. That is, we have not included in our architecture facilities for processing dynamic performance data – only the facilities we believe are necessary to collect and distribute it efficiently. We leave as an open question, hoping to stimulate discussion, the question of whether dynamic performance data and dynamically generated program events should be managed in the same way on the Grid.

Finally, we are not advocating any particular interface, or set of interfaces for accessing performance data on the Grid. It is possible to envision a single interface that permits access to dynamic, lifetime-limited performance data, archival data, and static "metadata." We believe that it is also possible to formulate arguments (based on performance concerns) for separate interfaces. Our hope is that these issues and others will be addressed as the Grid becomes more pervasive.

4 Conclusion

We believe that the requirements for dynamic performance data management on the Grid are sufficiently unique to warrant a coherent architecture for meeting them. In this document, we articulate our current observations of what those requirements are and a logical architecture that has emerged from our work for satisfying them. We do not contend that this architecture is either minimalist or unique. Rather, we present our rationale for its components as a way of stimulating discussion and further investigation.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[2] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, August 1997.

[3] J. Plank, M. Beck, and W. Elwasif. IBP: The internet backplane protocol. Technical Report UT-CS-99-426, University of Tennessee, 1999.

[4] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing 1998*, July 1998.

[5] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 1999. available from `http://www.cs.utk.edu/~rich/publications/nws-arch.ps.gz`.